ARM Cortex Family On-Chip Emulation

This document is intended to be used together with the CPU reference manual provided by the silicon vendor. This document assumes knowledge of the CPU functionality and the terminology and concepts defined and explained in the CPU reference manual. Basic knowledge of winIDEA is also necessary. This document deals with specifics and advanced details and it's not meant as a basic or introductory text.

Contents

1 Introduction

This document describes winIDEA configuration and options related to Cortex devices. From iSYSTEM documentation please also refer to hardware reference document shipped along the “blue box” hardware for information on how to set it up and use it.

User is also encouraged to get familiar with the documentation from ARM. Following are suggested documents:

- ARM® Debug Interface v5 Architecture Specification
- ARM® Debug Interface v5 Architecture Specification ADIv5.1 Supplement
- CoreSight™ Architecture Specification
- CoreSight™ Components Technical Reference Manual

Debug features:

- Hardware execution breakpoints
- Unlimited software breakpoints
- Hardware data access breakpoints
- Fast internal/external FLASH programming
- Multicore support
- Real-time memory access
- Little and big-endian support
- On-Chip Trace support
- Parallel, SWO and ETB trace supported (CPU dependent)
- Exception catching
- Stopping peripherals (e.g. TIMERs) when stopped (CPU dependent)
- JTAG and SWD debug protocol
- Periodic service for feeding watchdog

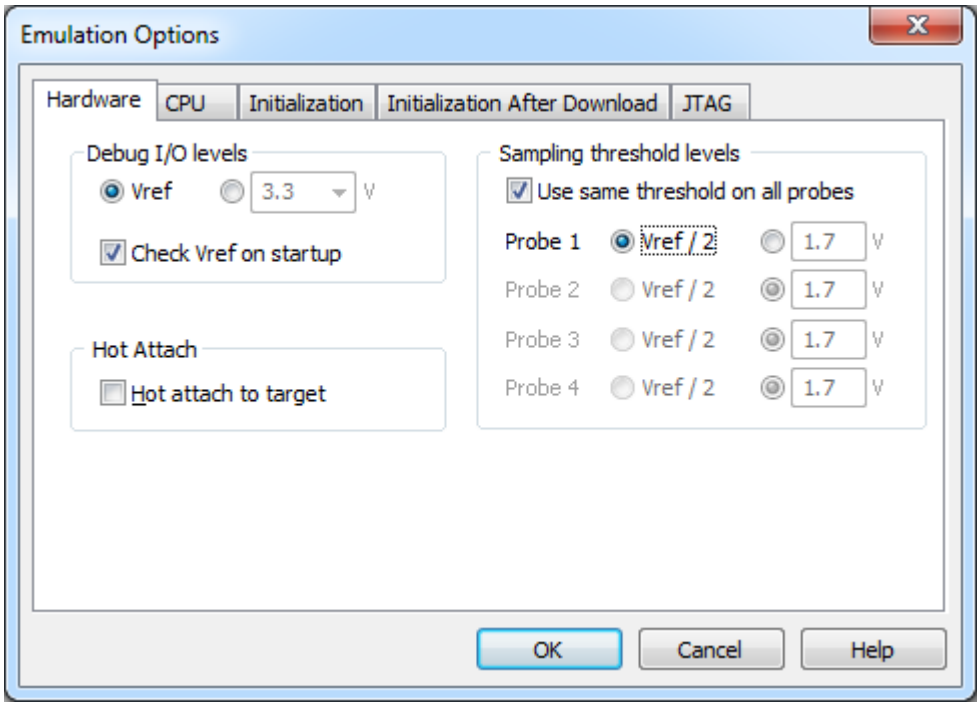
Supported CPUs

winIDEA supports various Cortex based microcontrollers (e.g.: Texas Instruments Stellaris, TMS and OMAP, ST STM32, NXP LPC, Freescale Kinetis and iMX, Xilinx Zynq etc.). Microcontrollers usually have on-board peripherals which are accessible via a memory mapped interface. For most commonly used microcontrollers winIDEA provides access to these peripheral registers via a Special Function Register (SFR) interface. SFR interface allows the user to view and modify peripheral registers using a visual tree structure with register names which are derived from microcontroller peripheral register specifications (refer to Special Function Registers technical notes for more information).

Check with iSYSTEM for the latest list of supported CPUs.

2 Emulation Options

2.1 Hardware Options



Emulation options, Hardware pane

Debug I/O levels

The development system can be configured in a way that the debug signals are driven by the emulator or by the target voltage (Vref).

When **'Vref'** is selected the voltage applied from the target to the reference voltage pin on the debug connector is used as a reference. It is fed to voltage follower which powers buffers driving the debug signals. The user must ensure that the target power supply is connected to the Vref pin on the debug connector and that it is switched on before the debug session is started. Initial debug connection will probably fail if these conditions are not met. However, it may succeed but the system behavior will be abnormal.

Alternatively emulator can force voltage for debug signals. Few preconfigured selections are possible from drop down menu. Desired value can also be set manually to other values.

Enable **'Check Vref on startup'** to perform reference voltage check on the emulators which have this feature.

Consult the specific emulator hardware reference for options availability.

Sampling threshold levels

These are settings for digital threshold levels on emulator systems with advanced debug connector probes. Up to four probes can be used for trace data acquisition. Each can have own threshold configure or all can use the same setting.

Hot attach

Hot attach is used for connecting physically disconnected emulator to already running target. Note that initialization sequences are skipped and reset method is not performed when Hot attach is enabled (refer to Initialization Sequence and Reset chapters for more information).

Procedure for using Hot attach is as follows:

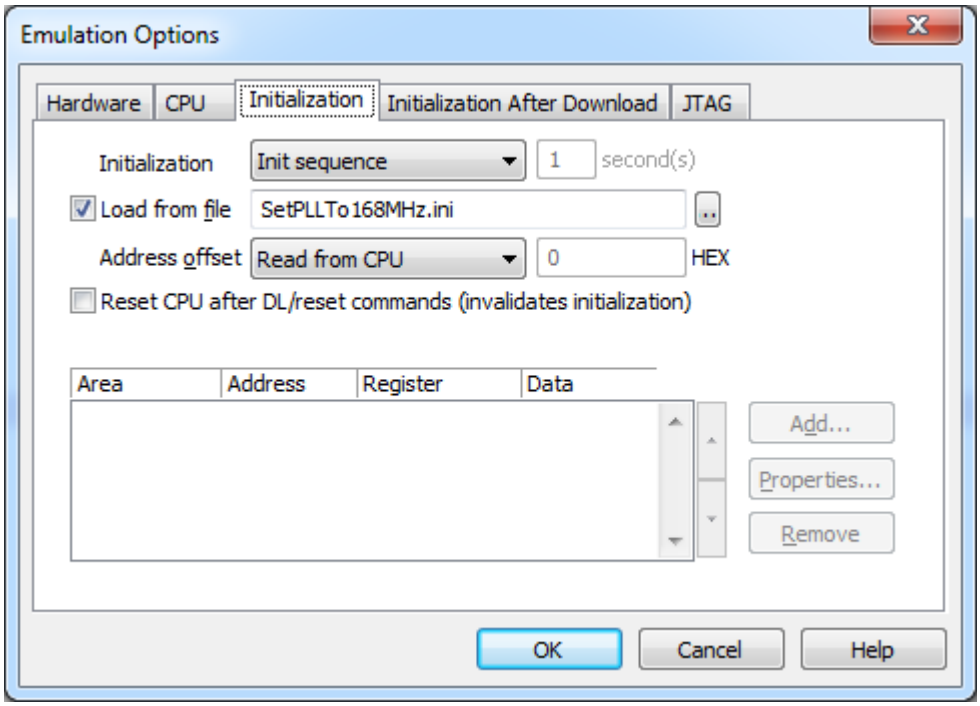
- Target is powered and running; emulator is physically disconnected from target and off
- Power on the emulator
- Enable **'Hot attach to target'**
- Execute **'Download'**, **'Download Symbols Only'** or debug **'Reset'**
- Emulator will initialize and put unneeded signals (e.g. SRST) to tristate
- Status will show ATTACH meaning that it is save to physically attach the debug cable to target
- Attach the debug cable
- Execute **'Attach'**
- winIDEA will connect to target without disrupting the execution

From there on the debug operations will behave as usual. E.g. stopping the target, setting breakpoints, resetting the target are possible.

2.2 Initialization Sequence

Initialization sequence is available for execution after reset method has been applied (refer to Reset chapter for more information) and target system is stopped under emulator control. A sequence is made from memory and/or register writes, delays and memory and/or register polls. Sequence is usually used to:

- Configure PLL and other system clock options (to enable faster debug clock and faster download)
- Initialize GPIOs and memory controller for external memory (to enable download to external memory)
- Disable watchdog
- Application specific settings; e.g. setting predefined memory location to custom magic value

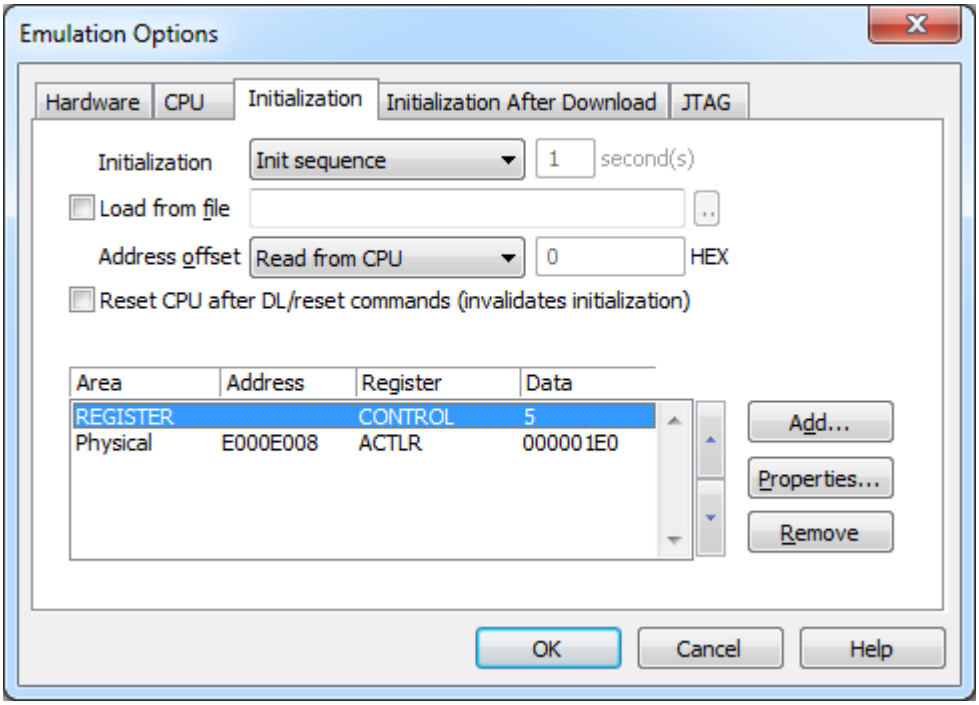


Emulation options, Initialization pane – initialization from file

Offset for accesses to Special Function Registers (refer to Special Function Registers technical notes for more information) can be specified when **'Address offset'** is set to **'Specify'** instead of **'Read from CPU'**. This information is becoming obsolete since modern devices have larger continuous address space.

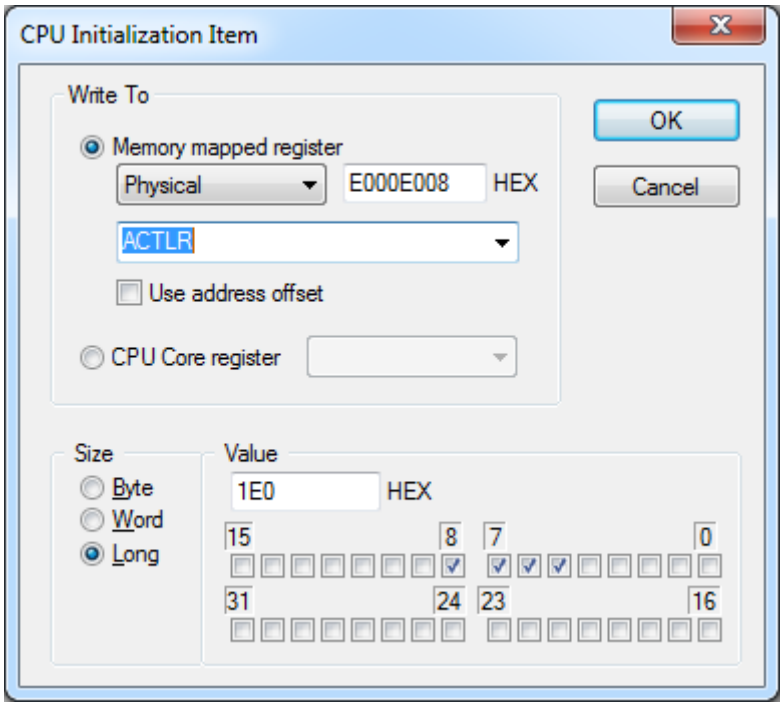
The initialization sequence can be set up in different ways:

1. As a text file with .ini extension. Refer to Initialization Sequence help topic for more information on syntax.
2. By adding necessary register and/or memory writes directly in the Initialization pane:



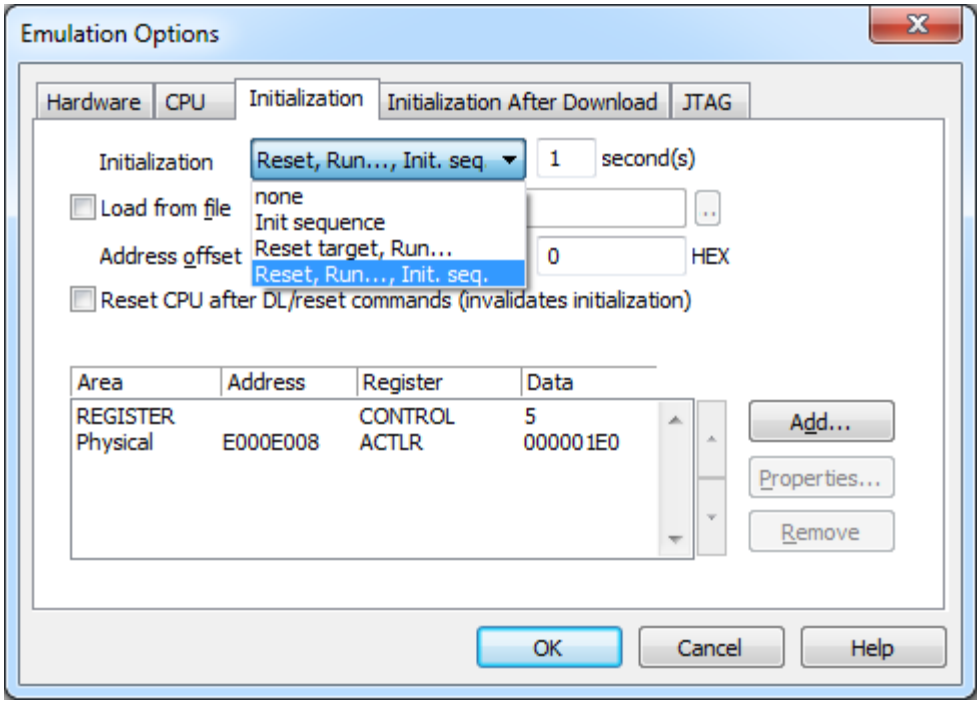
Emulation options, Initialization pane – direct write initialization

Adding and editing of the initialization items is done with **'CPU Initialization Item'** dialog where value is specified and destination of a write is selected. For memory writes user can select a special register address from the drop down list of special function register. Special Function Registers offset address can be added to address when **'Use address offset'** is enabled



Emulation options, Initialization pane, CPU initialization Item

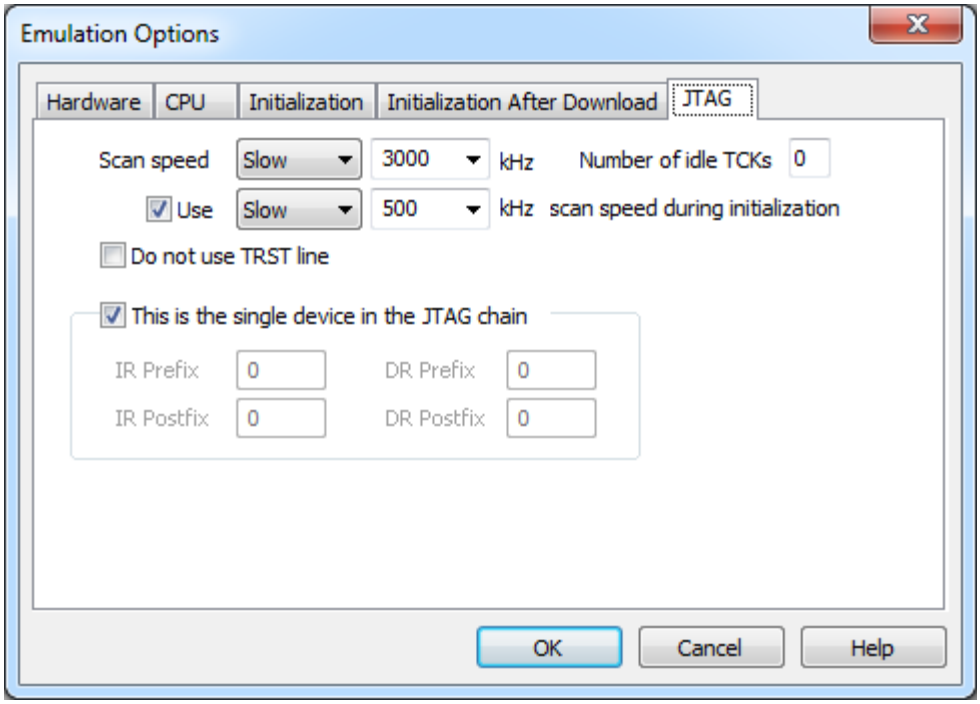
3. Allowing the preloaded firmware (e.g. BOOT in ROM) to initialize the target by selecting **'Reset target, Run…'**. One of previous memory access initialization method can follow when **'Reset, Run…, Init. Seq.'** is selected. Both ways leave the target to run for set count of **'seconds'**.



Emulation options, Initialization pane – run initialization with or without memory accesses

After initialization sequence completes the JTAG debug clock will switch from initialization speed to normal speed (refer to JTAG pane for more information) and download will start if emulation was started with **'Download'** action. Another initialization sequence can be specified on **'Initialization After Download'** pane to execute after download completes. It has same configuration options expect from **'Reset CPU after DL/reset commands'** which applies reset method (refer to Reset chapter for more information) after download before second initialization sequence.

2.3 JTAG



Emulation options, JTAG panel

Note: This tab is disabled when debug interface other than JTAG is selected.

Scan speed

The JTAG chain scanning speed can be set to:

- Slow - delays are introduced in the JTAG scanning to support the slower devices. JTAG clock frequency from 1 kHz to 4000 kHz can be set
- Fast – the JTAG chain is scanned with no delays
- Burst – provides the ability to set the JTAG clock frequency from 4 MHz to 100 MHz (DR scans are accelerated)
- Burst+ – provides the ability to set the JTAG clock frequency from 4 MHz to 100 MHz (repetitive JTAG scan cycles are accelerated)
- RTCK - Adaptive RTCK clocking for ARM
- Free – JTAG clock is always active (while there is no useful data to shift through the TAP it is held in IDLE state)

Speed mode availability depends on emulator used and target CPU. Also not all speeds will work with all targets. TAP scan frequency is usually bound to target CPU system clock and it is couple of times slower.

Number of idle TCKs

Define number of trailing TCK cycles after JTAG scan returns to Test-Logic-Idle JTAG state. Can be used to increase communication robustness at higher JTAG speeds.

Use – Scan Speed during Initialization

This option is useful on systems which start very slow. Slower scan speed can be used for first initialization sequence (see Initialization Sequence details), during which the CPU clock

is raised (PLL engaged) and then higher scan speeds can be used for download and later while debugging.

Do not use TRST line

Enable this when assertion of JTAG TRST line is not desired. TAP will be reset only by clocking through Test-Logic-Reset JTAG state.

JTAG chain

Enable **'This is the single device in the JTAG chain'** when only target CPU is connected to debug connector. Multiple TAPs inside target SoC are handled by winIDEA accordingly to CPU selection.

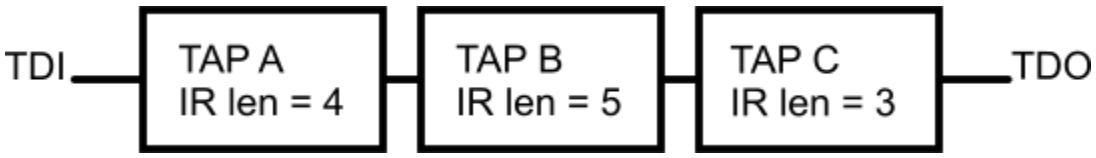
When JTAG chain contains multiple TAPs the IR and DR prefixes and postfixes must be properly configured:

- **'IR Prefix'** to the sum of IR lengths of all following* TAPs
- **'IR Postfix'** to the sum of IR lengths of all preceding* TAPs
- **'DR Prefix'** to the number of following* TAPs
- **'DR Postfix'** to the number of preceding* TAPs

winIDEA can help detect the connected TAPs. Please refer to information about Hardware Tools JTAG chain dialog. There winIDEA reports detected IR offsets of detected TAPs which are same as IR Prefixes or offset from TDO of last* TAP.

* In a topology sense in direction from TDI to TDO (pre/post terminology was chosen based on when filling bits need be shifted into TDI).

Example:

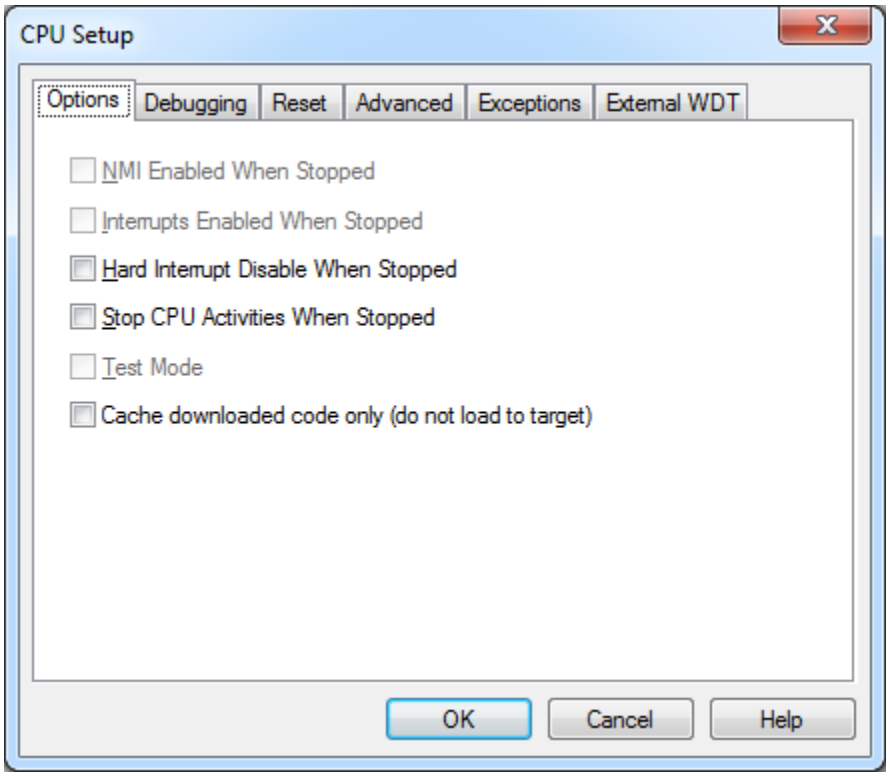


JTAG chain example

Target TAP Setting	Target CPU TAP is TAP A	Target CPU TAP is TAP B	Target CPU TAP is TAP C
IR Prefix	5 + 3 = 8	3	0
IR Postfix	0	4	4 + 5 = 9
DR Prefix	2	1	0
DR Postfix	0	1	2
detection IR offset	5 + 3 = 8	3	0

3 CPU Options

3.1 General Options



CPU Setup, Options pane

Hard Interrupt Disable When Stopped

When this option is checked interrupts will be enabled immediately after program execution resumes.
Otherwise, the CPU must execute a couple of instructions before returning to the program to determine whether interrupts were enabled when the CPU was stopped. These extra instruction executions can prevent task preemption when an interrupt is already pending.

Stop CPU Activities When Stopped

When the option is checked internal peripherals which can be stopped (e.g. timers) are stopped when the core is stopped. Usually when the option is checked the emulation system behaves more consistently while stepping through the program. While being aware of the consequences, it is up to the user whether the option is checked or not. E.g. it is recommended that a timer which generates interrupts is stopped when the application is stopped. Otherwise the CPU would first service all pending interrupts (generated by the timer while the application was stopped) after the application is resumed. Such behaviour is far away from the actual behaviour of the target application.

Note: This option is available for specific microcontroller families only.

Cache downloaded code only (do not load to target)

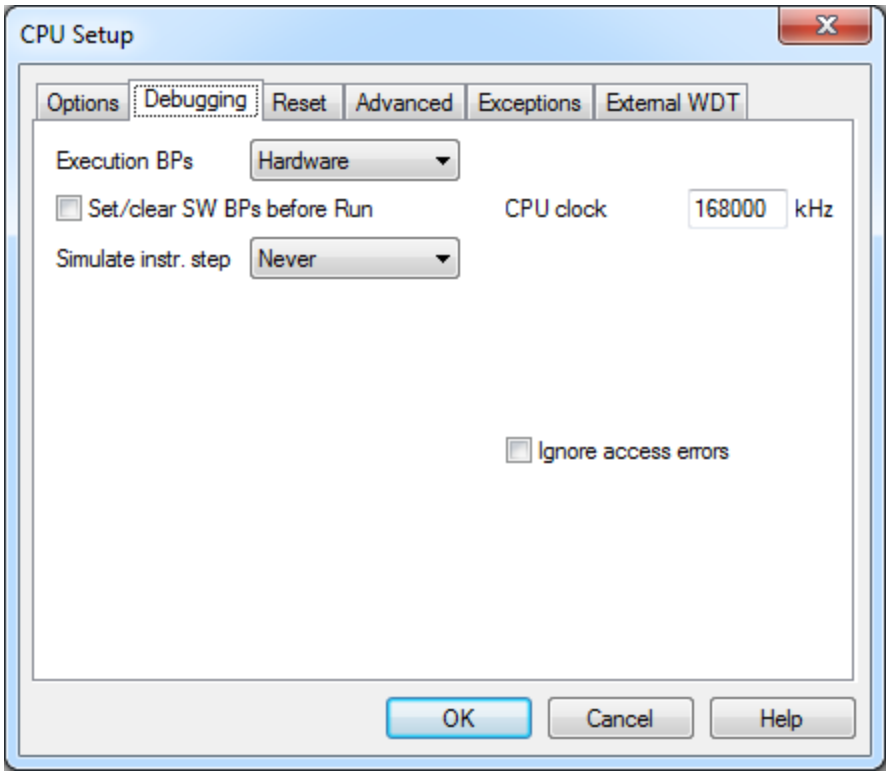
When this option is checked, the download files will not propagate to the target using standard debug download but the Target download files will.

In cases where the application is previously programmed in the target or it's programmed through the flash programming dialog the user may uncheck **'Load code'** in the **'Properties'** dialog when specifying the debug download file(s). By doing so the debugger loads only the

necessary debug information for high level debugging while it doesn't load any code. However, debug functionalities like trace will not work then since an exact code image of the executed code is required as a prerequisite for the correct trace program flow reconstruction. This applies also for the call stack on some CPU platforms. In such applications **'Load code'** option should remain checked and **'Cache downloaded code only (do not load to target)'** option checked instead. This will yield in debug information and code image loaded to the debugger but no memory writes will propagate to the target, which otherwise normally load the code to the target.

The option is deprecated but kept for back compatibility. When target memory(s) have the application already programmed user should perform **'Load Symbols Only'**. The operation is equal to **'Download'** operation only without physically programming the target memory(s).

3.2 Debugging Options



CPU Setup, Debugging pane

Execution Breakpoints

Hardware Breakpoints

Hardware breakpoints are breakpoints that are already provided by the CPU. The number of hardware breakpoints is limited per device.

Note that the debugger, when executing source step debug command uses one breakpoint. Hence when all available hardware breakpoints are used as execution breakpoints the debugger may fail to execute debug step. The debugger offers **'Reserve one breakpoint for high-level debugging'** option in the **'Debug/Debug Options/Debugging'** tab to circumvent this. By default this option is checked and the user can uncheck it anytime.

Software Breakpoints

The debugger can use unlimited software breakpoints to work around limited hardware breakpoints. Debugger uses dedicated software breakpoint instruction to implement software breakpoints. This instruction is overwritten over original data at breakpoint address. When

execution is continued from a location like that the original instruction must be written back. Single step is made first to execute the instruction at breakpoint location. Then original instruction is again replaced with breakpoint instruction and CPU is put into running.

When **'Set/clear SW BPs before Run'** is enabled all changed breakpoint locations are updated right before run. This is useful when using SW BPs in programmable memory (e.g. internal FLASH) since memory doesn't need to reprogrammed for each SW BP change (to change only couple of bytes in programmable memory whole erasable sector must be programmed).

Set/clear SW BPs before Run

When the option is checked, then a software breakpoint is not set / cleared immediately, but is just remembered. Only when the CPU is set to running are the breakpoints committed. This way several breakpoints can be changed but only one re-FLASH operation takes place. This is especially noticeable in testIDEA operation with many stubs and also during a regular debugging session when several breakpoints are set / cleared within the same flash erase block.

Simulate instr. step

'Never' is selected by default. When run or source step debug command is executed from a BP location the debugger first clears BP, executes single step, sets back the original BP and then resumes the application. All this is done in background hidden from the user. Since setting and clearing software flash breakpoint can be very time consuming the new approach was introduced. It simulates the first instruction at breakpoint address without clearing and setting the software flash breakpoint. Thereby the user can select 'FLASH SW BP' in order to speed up the debugging. Not all instructions can be simulated successfully. If the option yields erroneous behavior, set back to the default setting.

CPU clock

Set this to core clock. It might be used for asynchronous trace timing or internal flash programming timing, depending on the CPU that is being used. When it is used for both set to core clock valid when trace is active then use **initialization sequence** to set the CPU clock the same frequency for programming.

Note that this setting is critical to flash operations on many CPUs. If flash programming fails, check if this setting is correct. This setting must be valid at the moment when flash programming is performed. When the application is first downloaded the CPU runs on the default clock (out of reset). This means that the clock setting must be correct at the time of download. If you wish to use software breakpoints / write to flash once the CPU clock is changed (e.g. PLLs are used), then it is necessary to set this setting to the clock that is valid at this time and create a **winIDEA initialization script** that will set the CPU clock to the same frequency prior to application download. This ensures that the CPU clock setting is valid both at the time of the application download and later on as well. Refer to Initialization Sequence for more information.

Example (50MHz is entered as CPU clock):

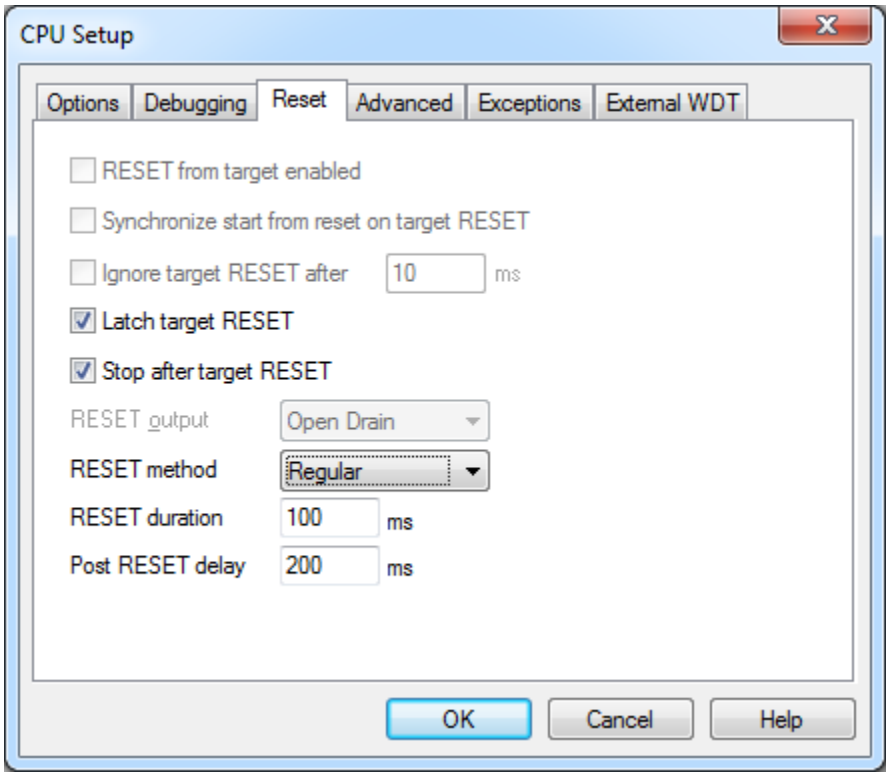
1. CPU runs on 12MHz out of reset (download will not work)
2. Initialization script sets the CPU clock to 50MHz
3. Download is performed
4. CPU is reset (clock is reset back to 12MHz) – if **Reset CPU after DL/reset** option is enabled
5. Flash operations / trace unoperable until the clock is set to 50MHz by the application

Note that this might not be necessary on all Cortex devices.

Ignore Access errors

When checked, the debugger identifies memory access errors for individual memory location(s). When the option is unchecked, the debugger would declare access error for remaining memory locations once one access error is detected within a memory read block, which is used in the disassembly window or memory window.

3.3 Reset



CPU Setup, Reset pane

Latch target RESET

System reset line drivers on debugger side and target side are open collector. Either can pull it low. Assertion from target side can be missed by debugger if it is shorter than poll period of reset line. When the option is checked (default) the debugger hardware latches the reset line when being pulled low by target system. When polled next time the winIDEA acknowledges reset state. This yields a delay from when line is asserted by the target reset and the acknowledgement by winIDEA and potential restart of application. An example is an application where the CPU is periodically set into a power save mode and then woken up by an external reset circuit. In such case a delay introduced by the debugger would yield application not operating properly.

Stop after target RESET

Enable to stop the CPU after reset is detected.

RESET method

Different reset methods are possible due to the problems arising from various wiring of the TRST (JTAG reset), the SRST (system/CPU reset) and possible other reset signals (e.g. POR - power on reset) in the target system or inside the SoC (System on Chip). It is recommended that these otherwise independent signals are not connected together. However, sometimes they are tied together which can conflict with the debugger controlling the target microcontroller.

Proper reset wiring on debug connector should have independent TRST and SRST signals. Also their functionality should be independent –i.e. target debug module is reset when only TRST is activated while rest of the system continues running. Respectively debug module is left intact while only system is reset via SRST signal. This functionality depends on silicon design and board design.

To keep the functionality needed for proper target initialization (explained further in text) other reset signals should be wired not to disrupt the described reset functionality. E.g. SRST if often replaced with POR signal on debug connector. Or sometimes SRST and POR are connected. POR usually resets debug module like TRST does which is not expected by emulation tools when putting system in reset state via SRST debug signal. Power on reset is one time event after powering on the system and should not repeat while power stays on.

Following are the desired steps for proper initialization when emulator connects to the target; (assuming target system is already powered on and debug connector is attached):

- target system is put in reset with activating SRST and not releasing it (this is emulation hardware initialization step also available as a separate **'Hardware/Initialize'** command where target system doesn't need to be powered on yet; following are target initialization steps),
- target debug module is reset: TRST is activated for a moment and then released,
- while system is still in reset a debug mechanism is set up to stop CPU execution when released from reset (also other debug settings are applied),
- SRST is released and core doesn't run. The target microcontroller is in known state and ready for next debug actions.

If needed to ensure that a CPU doesn't run after power on when using a regular reset method first initialize the emulation hardware with **'Hardware/Initialize'** command.

A default reset method is **'Regular'**. If it's not working try **'SoC'** (if available) and last **'Stop and preset'**.

When **a 'Regular'** method doesn't work it is recommended to put a short delay at the beginning of the application for debug purposes. This ensures that the CPU peripherals don't get configured before the debugger takes the control over the microcontroller. Nevertheless, keep in mind that CPU peripherals might not be in its reset state after the debug reset when not using a **'Regular'** reset method.

- **Regular**

  This reset method assumes that TRST and SRST signals are separately connected from the microcontroller to the target debug connector. When the debug session is initialized a debug module is reset first via the TRST while the microcontroller is held in reset via the SRST thus preventing the core to run. The SRST is released after the debug module has been configured to stop the core after SRST is released.

- **Stop and preset**

  Also called a software reset method because it does not drive any hardware signals. When debug session is initialized the debug module is reset and the system reset (SRST) is released. Core is then stopped with a debug command and its execution point is set to reset value by modifying the program counter. As with SoC method the system is left running from power on until the core is stopped.

  Note: With this method the core and peripherals are not reset on debug reset command.

- **SoC**

  SoC stands for System on Chip reset. This is a method based on a mechanism implemented in the CPU SoC. This mechanism enables to reset the core via a debug

command. It is implementation specific whether it also resets CPU peripherals. When the debug session is initialized the system reset is released and the debug module is reset. Next the debug module is configured to stop the core after the CPU is reset. Afterwards a reset command is issued to the debug module which yields resetting the core (and usually peripherals) and stopping it.

Note: From power-on reset the core and the application code are already running for a short period of time while the debug module is being configured for the first time.

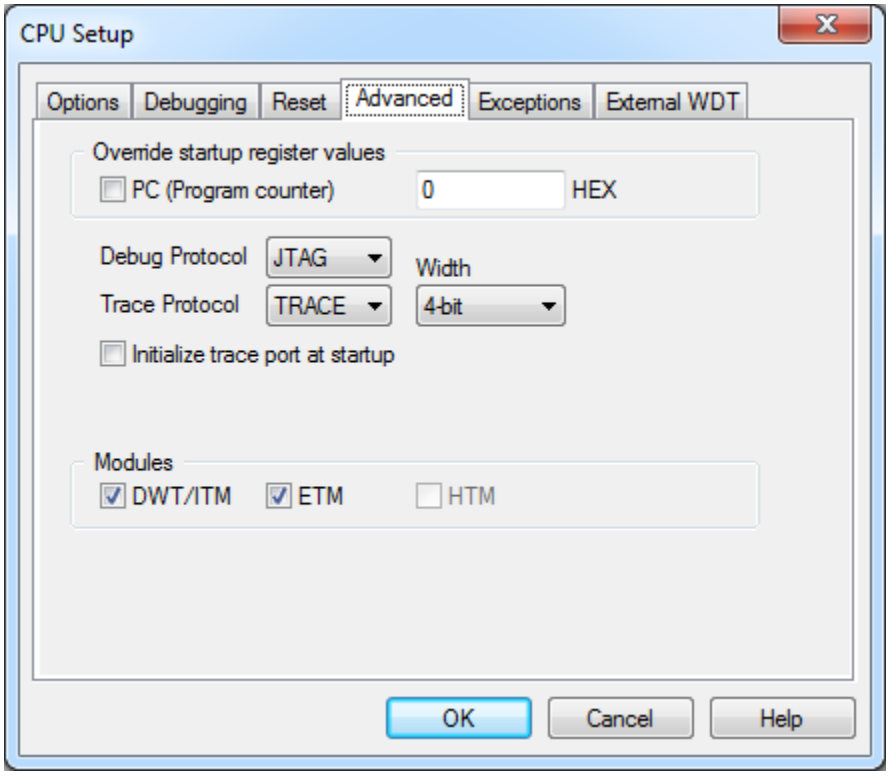
RESET Duration

The width of the RESET pulse is specified here.

Post RESET Delay

Typically the on-chip debug module is reset concurrently with the CPU. After the debugger releases the CPU reset line from the active state the on-chip debug module can require some time to become operational. This time can also depend on any additional reset circuitry on the target system. The default delay value normally allows the debugger to gain the control over the CPU. Try different delay values to establish the debug connection if a debug connection fails.

3.4 Advanced Options



CPU Setup, Advanced options

Override startup register values

This option overrides the Program Counter reset value with the value set.

Debug Protocol

Two protocols for debug are available:

- Serial Wire debug port (SWD)
- JTAG debug port (JTAG)

A microcontroller can support any or both protocols. Refer to microcontroller vendor documentation for details on debug protocol availability.

Note that debug connector doesn't reflect debug protocol. See truth table on debug, trace and connector combinations at the end of the chapter.

Trace Protocol

A microcontroller can support any or all protocols. Refer to microcontroller vendor documentation for details on trace protocol availability. Four protocols for trace are available:

- Asynchronous mode (SWO) – available only in combination with SWD

  With **'SWO'** selected the asynchronous clock prescaler must also be specified (0-FFFF hex). Trace clock for SWO is derived from the asynchronous reference clock using following formula:

  SWO output clock = asynchronous_reference_clock / (1 + prescaler value)

  Asynchronous reference (asynchronous_reference_clock) is the CPU clock which is specified in the **CPU Setup\Debugging\CPU** clock option.

  Note: Depending on the CPU clock rate and the quality of the trace line it may be necessary to use a SWO prescaler value greater than 0 in order to get the trace signal into the frequency range where it can be reliably recorded. Higher prescaler value causes the trace port output data rate to drop which may result in trace protocol overflows.

  Note: winIDEA configures SWO to operate in NRZ(UART) mode which requires accurate CPU clock. If CPU clock is not accurate enough the SWO trace recording might not work. Internal oscillators on some CPUs exhibit higher levels of inaccuracy which prevents a successful SWO trace recording in UART mode. Better results are achieved with CPU clock configured to use an external precise oscillator.

- Synchronous mode (TRACE) – available with JTAG and SWD

  The synchronous mode requires from 2 to 5 extra pins depending on the data trace size. In addition it is available with JTAG and SWD debug protocol and provides better bandwidth output capabilities than asynchronous trace where trace overflows are more common. Another advantage of the synchronous trace is the exact time information which makes profiler results trustworthy. Trace output is one of TPIU where trace source streams from multiple trace sources are formatted into single output trace stream.

  For **'TRACE'** protocol, the parallel port width must be selected (1-4 pins) depending on the target microcontroller.

- Embedded Trace Buffer (ETB) – available with JTAG and SWD

  No external pins are needed. Trace data is recorded to the internal dedicated memory and read through the debug channel. All produced trace data can be acquired without speed restrictions. Drawback of using the ETB is buffer size limit. Internal data buffer is of circular type. Trace can be recorded form CPU run until full or until CPU stop with history limited by buffer size. Recording or CPU need to be stopped to read recorded data which restricts the usage of reading the ETB while recording in real time.

- Legacy Embedded Trace Macrocell (ETM)

  For devices that have ETM module output pins available on the package.

Initialize trace port at startup

On some CPUs trace port cannot be initialized via the initialization sequence (.ini) or in run time and has to be done by the debugger immediately after getting control over CPU. Configuration mainly configures the IO ports/pins for parallel trace (TRACE). This includes

setting alternate function, direction, drive strength etc. for the TRACE IO pins. Not all devices need this option.

Same configuration is retried on each analyzer session start. This option is not needed if the configuration is possible after the debug session is active and the CPU is running for some time. It might be useful when analyzer session is started after the application firmware locks the access to IO configuration register. Note that firmware should not corrupt the IO configuration settings.

Modules

Configure which modules are available in debugged microcontroller when generic CPU is selected.

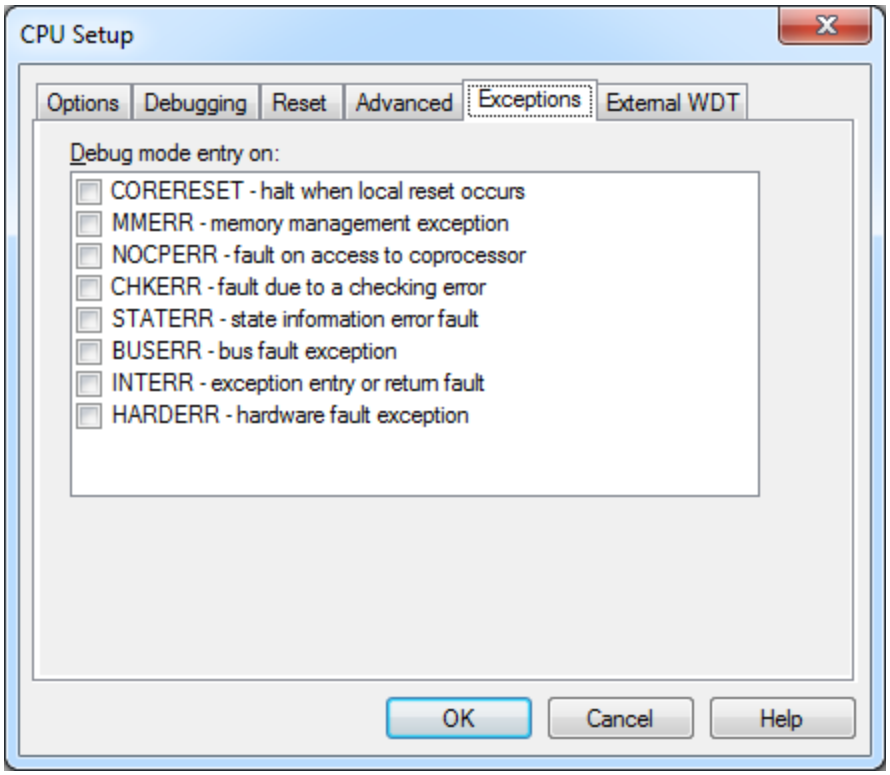
Debug connectors

Not all debug protocol, trace protocol and debug connector combinations are possible. Debug protocol is available unrelated to the used connector. Following table lists available trace protocols regarding to chosen debug protocol and debug connector.

Connector Debug protocol	Cortex 20-pin	Cortex 10-pin	ARM-JTAG 20-pin
JTAG	ETB TRACE	ETB	ETB
SWD	SWO ETB TRACE	SWO ETB	SWO ETB

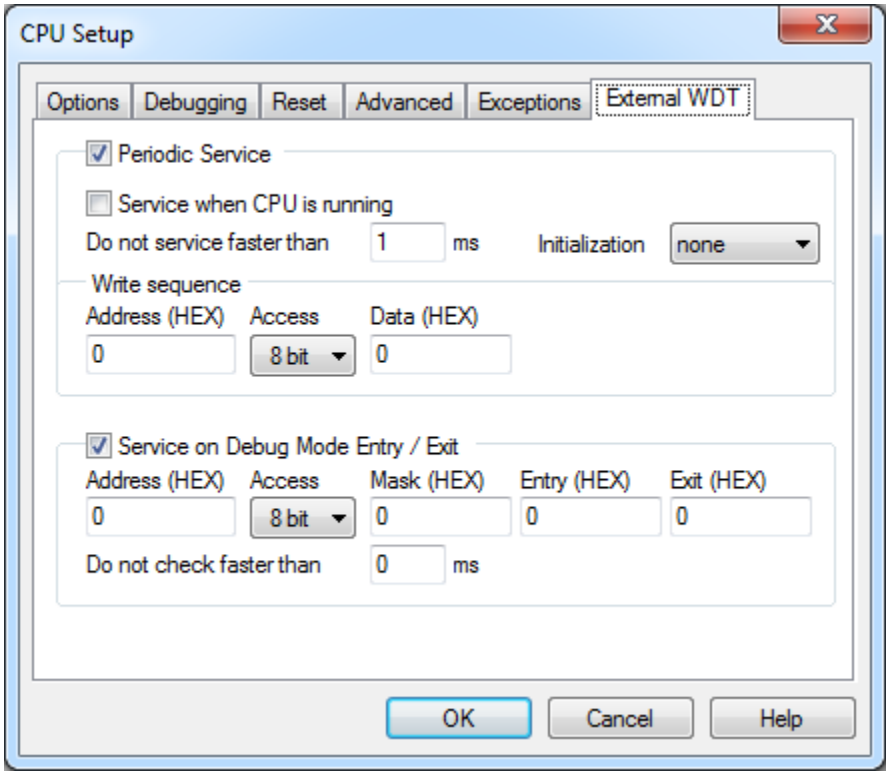
3.5 Exceptions

Debug module offers the configuration for entering debug mode on exception. The core can be halted on the selected exception vectors. It is recommended to select the exceptions which are not handled in the application. By doing so the application stops on such unexpected event.



CPU setup, Exceptions pane

3.6 External WDT



CPU Setup, External WDT pane

Periodic service and/or service on debug entry/exit can be configured to prevent the watchdog to disrupt the CPU while stopped.

Periodic service

- **Service when CPU is running**
  When enabled the emulator will also make configured memory writes if CPU is running.
- **Do not service faster than**
  Configured memory access period will not be less than this option. Set this to as high as possible to conserve debug communication for other debug operations.
- **Initialization**
  Not applicable. Select none except instructed otherwise from iSYSTEM support.
- **Address**
  Specify write address.
- **Access**
  Specify write size.
- **Data**
  Specify write value.

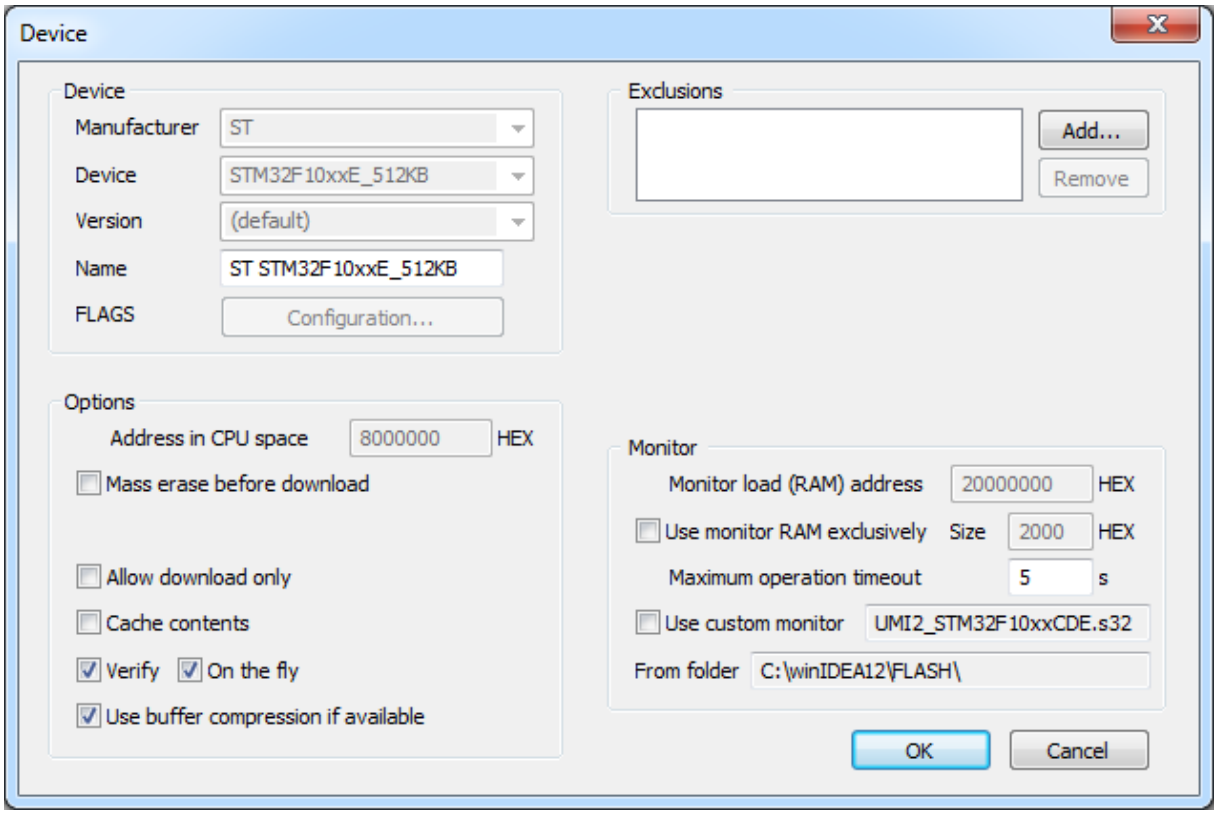
Service on Debug Mode Entry/Exit

- **Address**
  Specify write address.
- **Access**
  Specify write size.
- **Mask**
  Specify valid bits mask for read-modify-write.
- **Entry**
  Specify write value for debug entry.
- **Exit**
  Specify write value for debug exit.
- Do not check faster than
  Run/Stop status change will not be checked faster than specified by this setting. Set this to as high as possible to conserve debug communication for other debug operations.

4 Internal Flash Programming

Note: Specific flash related configuration and settings are described in the chapter Cortex-M and Cortex-A & Cortex-R, depending on the target architecture.

iSYSTEM tools support internal flash programming for devices that can be selected from **'CPU variant'** in Emulation Options, CPU pane. Hardware menu entry is added for selected device. Entry features available flash operations (e.g. Mass erase) and **'Configure…'** option to configure internal flash programming.



Internal flash programming options

Check the **'Erase before download'** option when a complete Program Flash must be erased prior to the programming. Only sectors where the code is loaded are erased when the option is unchecked.

It is recommend to check the **'Verify'** and **'On the fly'** options. With both options checked the flash programming monitor will (while still having the data to be programmed in the monitor buffer) verify if the input data was successfully programmed at the end of the programming cycle. This verify is much faster comparing to the standard Debug/Verify Download debug command which is then no longer required if download only targets internal flash.

'Use buffer compression if available' is another optimization feature. The data to be downloaded is compressed before sending to the flash programming monitor running on device and extracted there.

5 Cortex-M

For more information on Cortex-M specific configuration and settings refer to winIDEA Contents Help (or alternatively to the belonging standalone pdf document) describing Cortex-M On-Chip Emulation.

6 Cortex-A and Cortex-R

For more information on Cortex-A and Cortex-R specific configuration and settings refer to winIDEA Contents Help (or alternatively to the belonging standalone pdf document)

describing Cortex-A/R On-Chip Emulation.

7 Trace

Refer to winIDEA Contents Help, Analyzer Window section (or alternatively to the standalone Analyzer.pdf document) for general information on Trace user interface and use.

For more information on Cortex specific trace configuration and use refer to winIDEA Contents Help (or alternatively to the belonging standalone pdf document) describing ARM Cortex On-Chip trace in details.

8 Profiler

Refer to winIDEA Contents Help, Profiler Concepts section for Profiler theory and background.

Refer to winIDEA Contents Help, Analyzer Window section (or alternatively to the standalone Analyzer.pdf document) for information on Profiler user interface and use.

9 Coverage

Refer to winIDEA Contents Help, Coverage Concepts section for Coverage theory and background.

Refer to winIDEA Contents Help, Analyzer Window section (or alternatively to the standalone Analyzer.pdf document) for information on Coverage user interface and use.

10 Hardware Tools

10.1 Apply Target RESET

CPU reset resets the core, as well as the debug module. If you wish to reset only the core (to simulate external reset of the core), then choose **Hardware** / **Apply Target RESET** option. This option is available only when the target is running. The SRST line will be used to signal the reset. The duration of the reset signal (**RESET duration**) is set in the **CPU Options** / **Reset** dialog.

10.2 JTAG Scan

Note: This tab is disabled when SWD debug interface is used.

Open **Hardware** / **Tools** to access this functionality. This functionality allows the user to have access to the JTAG chain to which the debugger is connected in order to control the debugged CPU. Primarily it was designed for troubleshooting.

Operation:

Scan IR and return to Run-Test-Idle: starts instruction scanning in current state and returns to Run-Test-Idle state.

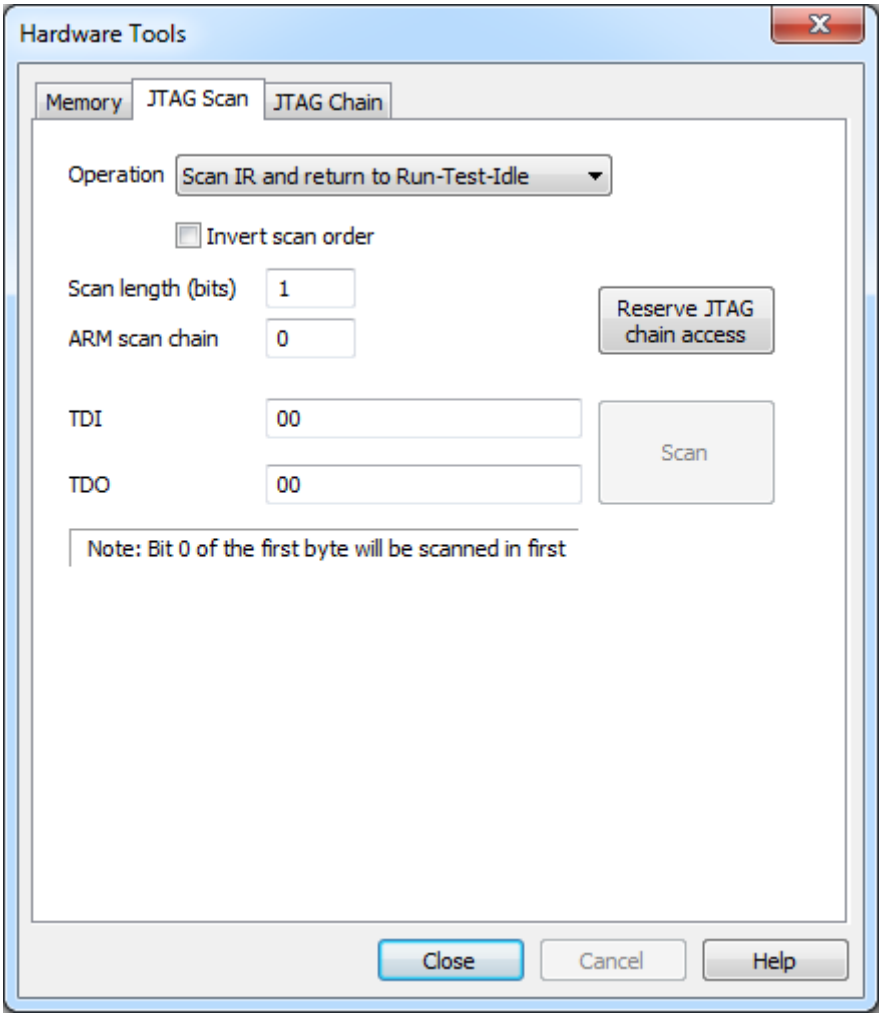
Scan DR and return to Run-Test-Idle: starts data scanning in current state and returns to Run-Test-Idle state.

Scan IR and return to Select-DR-State: starts instruction scanning in current state and returns to Select-DR-State state.

Scan DR and return to Select-DR-State: starts data scanning in current state and returns to Select-DR-State state.

Scan IR and return to Pause: starts instruction scanning in current state and returns to Pause state.

Scan DR and return to Pause: starts data scanning in current state and returns to Pause state.



Invert scan order

The data under “TDI” (DR scan only) can be scanned in both orders. If this option is not checked, then bit 0 (LSB bit) of first byte is scanned first. If this option is checked, then the bit pointed by “Scan length (bits)-1” is scanned first.

Example: TDI: 12345, Invert scan order [ ], Scan length = 16 bits… Bit stream scanned (bit on the left side scanned first): 0 1 0 0 1 0 0 0 0 0 1 0 1 1 0 0

Example: TDI: 12345, Invert scan order [x] , Scan length = 16 bits … Bit stream scanned (bit on the left side scanned first): 0 0 0 1 0 0 1 0 0 0 1 1 0 1 0 0

Scan length (bits)

The number of bits scanned at DR or IR scan.

ARM scan chain

Prior every DR scan the scan chain is set to this value.

TDI

DR/IR scan input bits

TDO

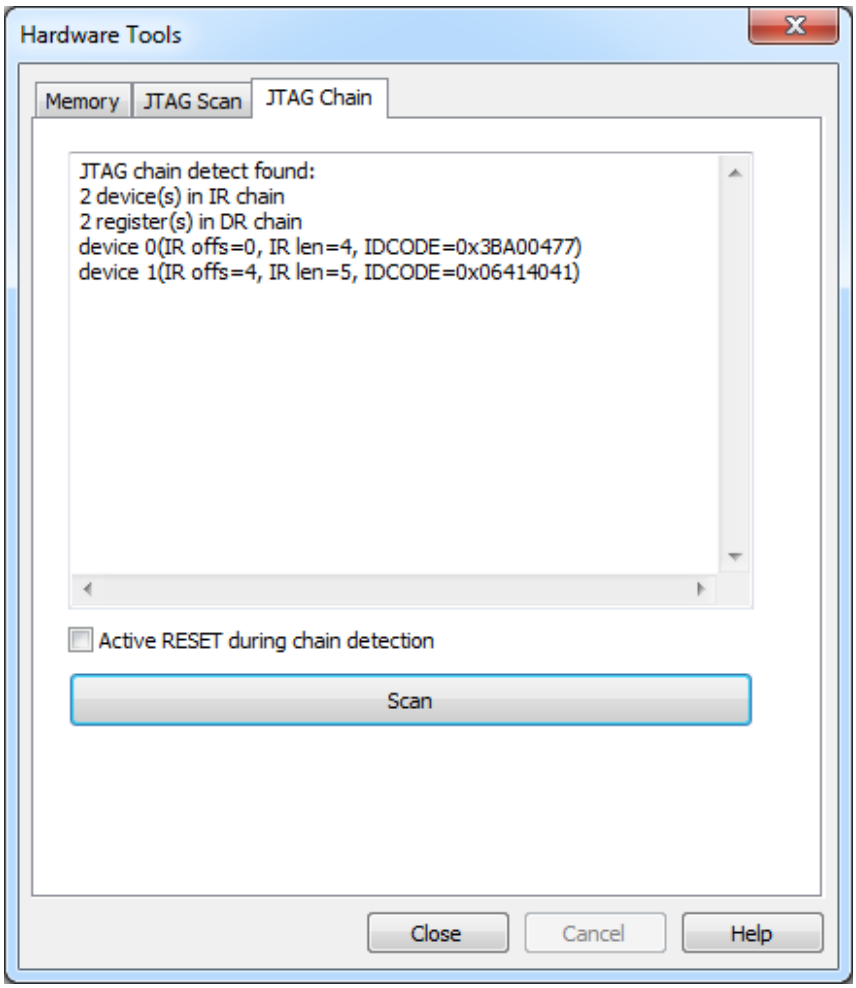
DR/IR scan output bits

Reserve JTAG chain access

When this button is pressed, only the scans through this dialog will be allowed (debugger will be “quiet”)

10.3 JTAG Chain

Open **Hardware** / **Tools** to access this functionality. This tool was made for JTAG connection test and device detection. Dialog will list detected JTAG TAPs (devices) after executing **‘Scan’**. Information can be used to configure JTAG setting (see Hardware Emulation JTAG options for more information). Additionally the reset line connection (see CPU options Reset for more information) can be determined from two scan results depending on **‘Active RESET during chain detection’**. SRTS and TRST are probably connected when scan finds TAPs without the option enabled and doesn’t find any with option disabled.



JTAG Chain detection

11 Troubleshooting

This section addresses issues which are in common for all Cortex devices.

Refer to the Cortex-M troubleshooting section for issues related to Cortex-M architecture only.

Refer to the general troubleshooting section for tips and hints on problems, which are architecture independent.

11.1 Error 304: Check debug adapter.

A Cortex-M based target can provide 20-pin or 10-pin target debug connector, through which the debug tool connects to the target microcontroller. Pay attention when 20-pin 2.54mm connector is provided. It can have either Cortex pinout or ARM pinout. Make sure you correctly identify the pinout and use according adapter, which comes along the debug tool. Both pinouts can be found in the iC5000 hardware reference. If a wrong adapter is used, the debug connection will fail and also damage to the hardware can occur.

Cortex-R and Cortex-A based targets feature 20-pin 2.54 mm target debug connector with ARM pinout. An adapter with Cortex pinout must not be used with these targets.

11.2 Debug connection fails or the application doesn't stop after reset

On Cortex devices microcontroller internal reset logic can be implemented in a various way. The debug tool must be aware of the reset logic implementation in order to connect to the target microcontroller and gain control over the application. The debugger can fail to connect to the target microcontroller or to stop the application after releasing the reset when incorrect reset method is selected in the 'Hardware/Emulation Options/CPU Setup…/Reset' tab. Refer to Reset section for more details on supported reset methods.